A Code Size Microbenchmark for C

Yang Chen, John Regehr
School of Computing, University of Utah
{chenyang, regehr}@cs.utah.edu

http://embed.cs.utah.edu/embarrassing

Test Input

```
int pmat (int m, int n, double *y)
{
  int i, j, k;
  k = 0;
  i = 0;
  while (i < m) {
    j = 0;
    while (j < n) {
      k++;
      j++;
    }
    i++;
  }
  return (0);
}
```

Pure function!

LLVM-GCC Output

```
Disassembly of section .text:

00000000 <pmat>:
0: 8b 44 24 04 mov 0x4(%esp),%eax
4: 85 c0 test %eax,%eax
6: 7e 0f jle 17 <pmat+0x17>
8: 8b 4c 24 08 mov 0x8(%esp),%ecx
c: 85 c9 test %ecx,%ecx
e: 7f 07 jg 17 <pmat+0x17>
10: 31 c9 xor %ecx,%ecx
12: 41 inc %ecx
13: 39 c1 cmp %eax,%ecx
15: 7c fb jl 12 <pmat+0x12>
17: 31 c0 xor %eax,%eax
19: c3 ret
```

Code Size: 26 bytes

GCC Output

```
Disassembly of section .text:

00000000 <pmat>:
0: 31 c0 xor %eax,%eax
2: c3 ret
```

Code Size: 3 bytes

Factor: 767%

Compilation Options

```
llvm-gcc -Os -mfpmath=sse
-mssse3 -march=core2
-mtune=core2 -fomit-frame-
pointer -fno-stack-protector -w
-c pmat.c
```

```
gcc -Os -mfpmath=sse -mssse3
-march=core2 -mtune=core2
-fno-stack-protector -fomit-
frame-pointer -w -c pmat.c
```

Motivation

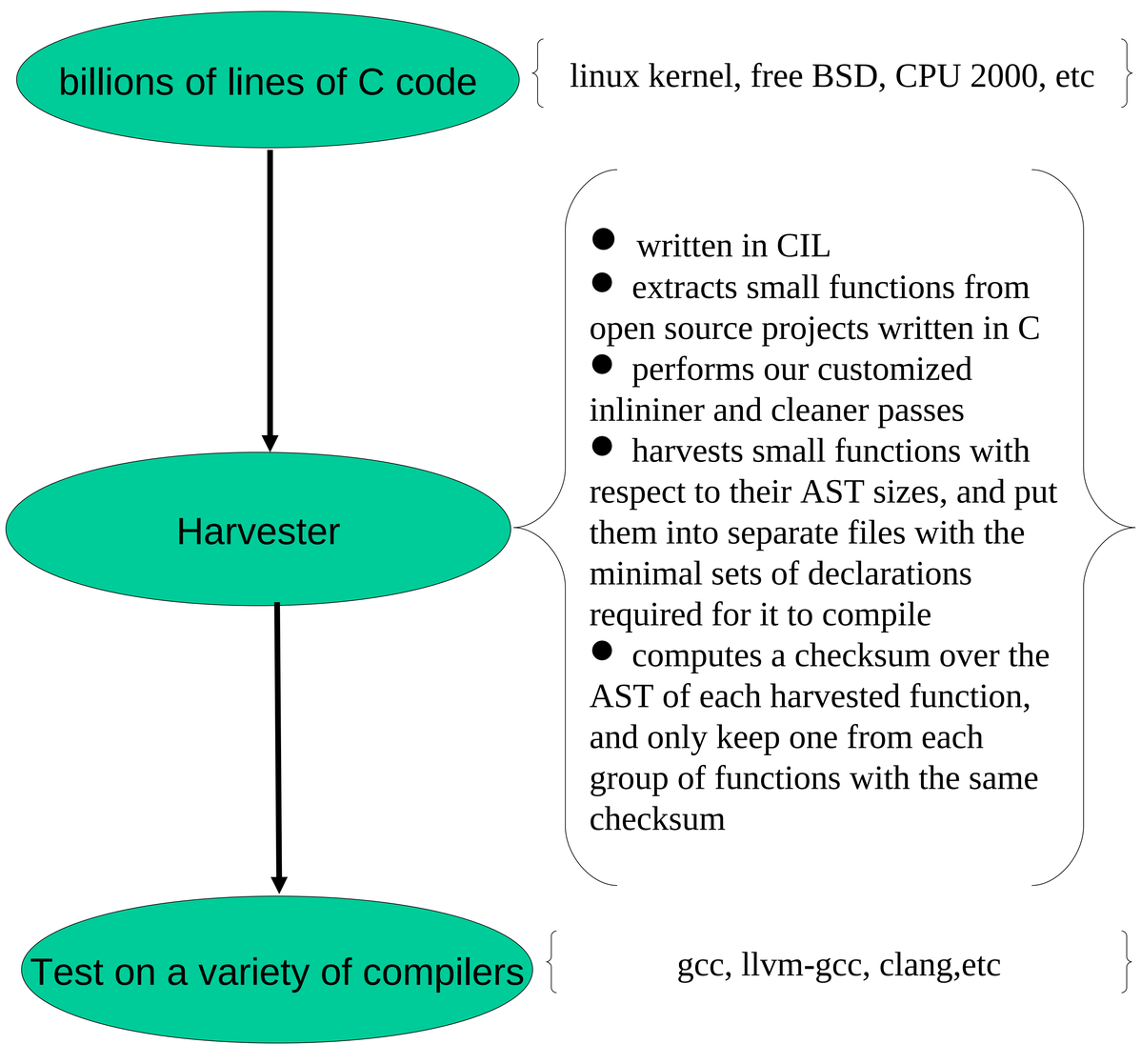
- No compilers could always generate smaller code than others
- Hand-optimized code is often hard to understand
- Programmers tend to write readable code and trust compilers generate fast and compact code for them
- There is a plenty of room for improving compiler optimizations

Reality to Compilers

- Compilers need tradeoff between the degree of optimizations and the amount of time spent in optimizing code
- Compilers suffer phase-ordering problems where the quality of generated code depends on the order of executed optimizations
- Compilers have to balance the machine-dependent and machine-independent features

Our goal: help compiler developers improve their products by giving them actionable test cases that pinpoint missed optimizations

Microbenchmark



Experimental Results

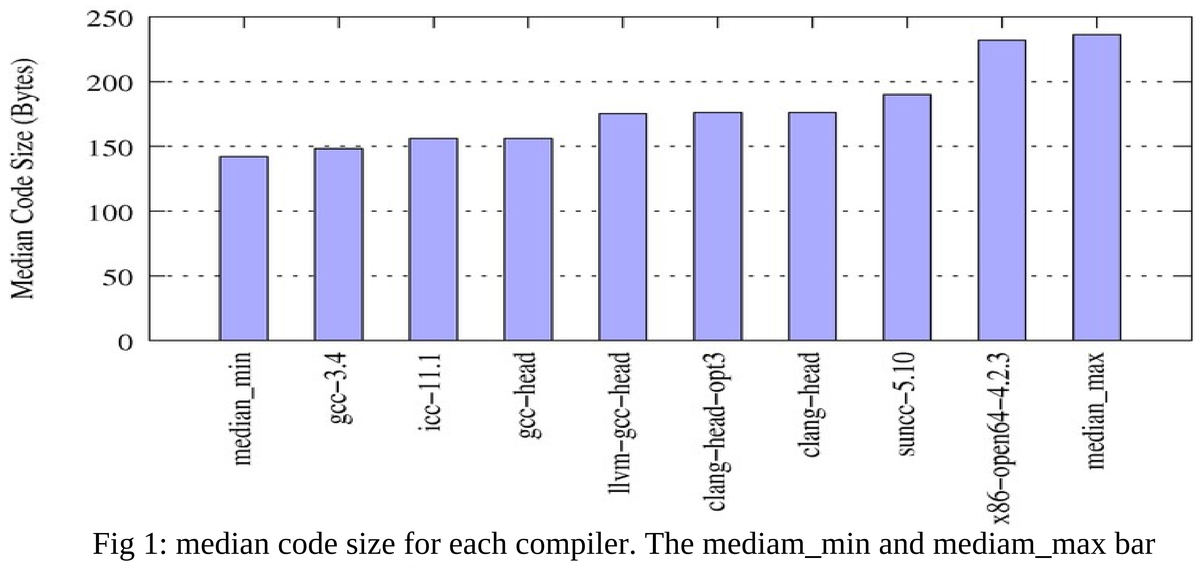


Fig 1: median code size for each compiler. The mediam_min and mediam_max bar shows the median size of the smallest and largest code produced by any compiler, respectively.

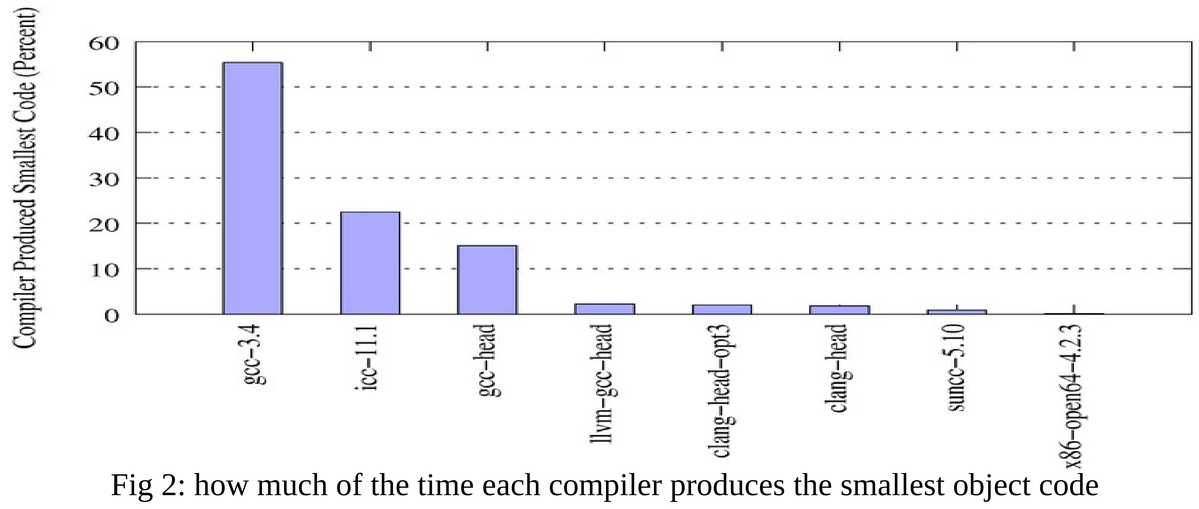


Fig 2: how much of the time each compiler produces the smallest object code